ECSE 425 Group 9 FSM Report

Joseph Cotnareanu, 260838160, Christian Martel, 260867191, Sam Perreault, 260829298

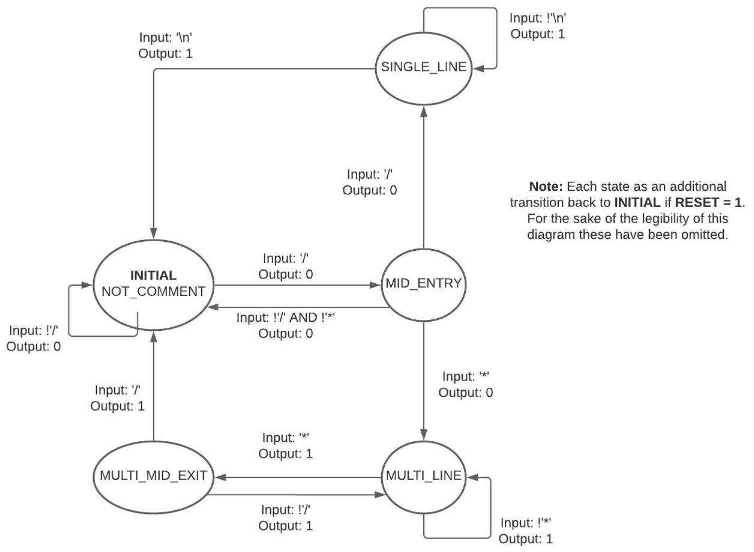
I. Introduction

We were asked to design a finite-state machine component able to identify the comments in a C code, to write the vhdl code for the component, and to write a complete vhdl test bench to ensure the fsm has the proper behaviour.

II. Design and Implementation

We designed Mealy finite-state-machine in which the output depends on the input and on the current state. The fsm must output '1' for characters that are part of a comment and '0' for characters that are not part of a comment. It's important to note that comment opening sequences, '/*' and '//' , are not considered as part of a comment and comment closing sequences, '*/' and '\n' are considered as part of a comment. Our design has the following five states:

- **Not Comment**: This state is the initial state. It indicates that the fsm is currently not in a comment section. The output is always '0'.
- **Mid-Entry**: This state indicates that the fsm is currently in the middle of the entering procedure of a comment. The fsm transits to this state once the forward slash character is inputted from the not-comment state. The output is always '0'.
- **Single-Line**: This state indicates that the fsm is currently in a single-line comment section. The output is always '1'.
- **Multi-Line**: This state indicates that the fsm is currently in a multi-line comment section. The output is always '1'.
- **Multi-Mid-Exit**: This state indicates that the fsm is currently in the middle of the exit procedure of a multi-line comment section. The fsm transits to this state once the star character is inputted from the multi-line comment state. The output is always '1'.



Note: Each state as an additional transition back to **INITIAL** if **RESET = 1**. For the sake of the legibility of this diagram these have been omitted.

Fig. 1. Our Mealy Finite State Machine

The state transitions and outputs for each state depending on the inputted character can be observed in **Figure 1**. It is important to note that each state transits to the initial not-comment state when the fsm is reset. These transitions were omitted from **Figure 1** for the sake of legibility.

The most complex aspect of this system is that some events must specifically occur consecutively (e.g. a slash followed by another slash or an asterisk to enter a comment, an asterisk followed by a slash to exit a comment). Thus, we have a mid-entry and mid-exit (where applicable) state that indicates that the first half the consecutive sequence of two characters has been seen. If on the very next input, the second half of the required sequence is not seen, we return to the state preceding the mid-state, and must wait again to enter mid-state. For example, say we are in NOT-COMMENT state. The input is a '/', now we are in MID-ENTRY state. There are two comment entry procedures and both are pairs of consecutive characters: '//' or '/*'. Therefore, the very next entry must be a '/' to enter a single line comment and enter the SINGLE-LINE state, or '*' to enter the MULTI-LINE state. If the actual input is, say, 'a', we move back to the initial state NOT-COMMENT and wait to enter the entry procedure again.

The vhdl implementation of the fsm component is available in the *fsm.vhd* file. The component has three input ports: a synchronous active-high clock single bit signal, an asynchronous active-high reset single bit signal and a 8-bit vector signal corresponding to the ASCII character inputted. The component has one output port single bit signal. The output is '0' when the input character is not part of a comment and '1' when the input character is part of a comment.

We defined a new signal type, *state_type*, to which we can assign five possible values (*NOT_COMMENT, MID_ENTRY, SINGLE_LINE, MULTI_LINE, MULTI_MID_EXIT*) corresponding to our five states. We added a *current_state* internal signal to keep track of the current state of the fsm. Three constant 8-bit logic vectors are used store the ASCII value of the key characters in our fsm implementation: '/', '*' and '\n'.

We defined a process with the clock and the reset input ports signals in its sensitivity list. The reset signal is asynchronous active-high. Hence, as soon as it is set to '1', the current state is set to the initial state *NOT_COMMENT* and to output is set to '0'. The reset signal has priority over the clock signal. The clock signal is synchronous active-high. Thus, when the reset is low, at a rising edge event of the clock, the current state block of the process is executed. In the state blocks, depending on the input vector and the current state, the appropriate output is assigned and then, the appropriate state transition is executed . For example, suppose the current state is *NOT_COMMENT* and the input is "00101111", the ASCII 8-bit vector representing the '/' character. On the next rising edge of the clock signal, the *NOT_COMMENT* state block will be executed. The output signal will be set to '0' and the current state signal will be set to *MID_ENTRY*.

III. Validation

To test and validate the test machine, it was decided to go for an "all edges" approach. This means that the test would be designed to traverse all of the transitions of the state machine. As these covered most, if not all of the edge cases possible, this test scheme is intended to be comprehensive.

Tests were run in EDA Playground, and as a result did not use the fsm_tb.tcl file that was provided.

These results indicate that our finite state machine is capable of detecting when the source code is in a comment, and whether the source code is entering and exiting comments. These results also indicate that the state can be easily reset with the reset signal, and that edge cases (such as ignoring a '*/' sequence while in a single line comment, as seen in figure 3) are properly addressed.

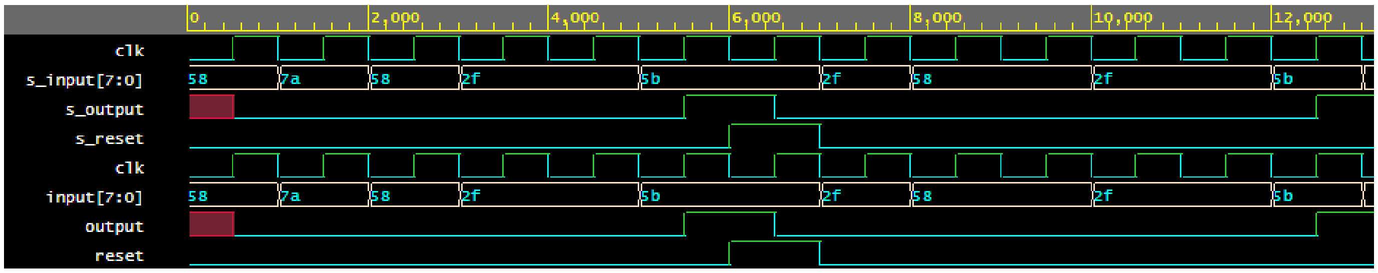


Fig. 2. Testbench results, including testing reset, single line comment. In hex, 2f = '/', 2a = '*', and a = '\n'

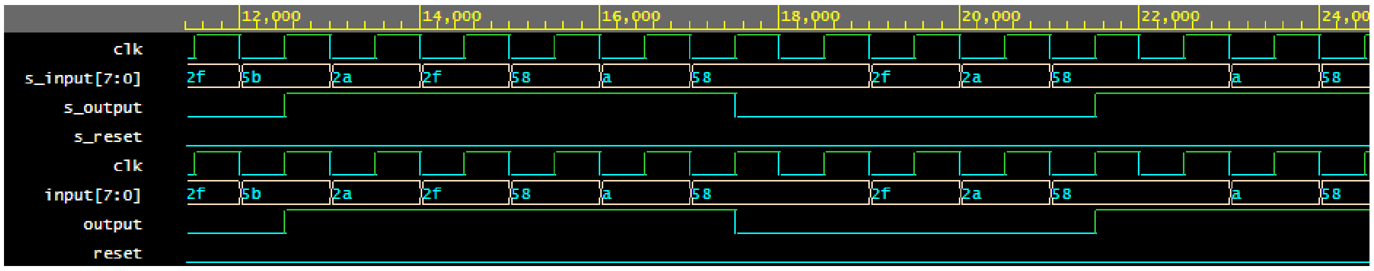


Fig. 3. Testbench results continued, including single line comment, multi-line comment. In hex, 2f = '/', 2a = '*', and a = '\n'

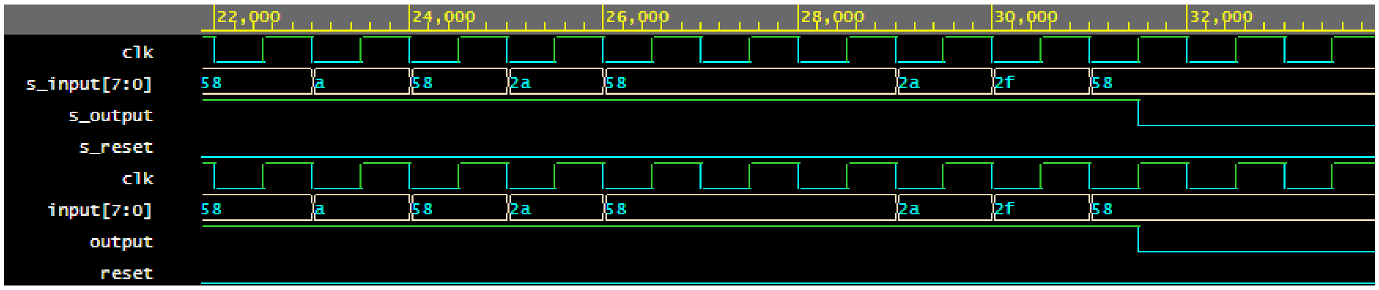


Fig. 4. Testbench results continued, including multi-line comment. In hex, 2f = '/', 2a = '*', and a = '\n'